ISTANBUL TECHNICAL UNIVERSITY
FACULTY OF ELECTRICAL AND ELECTRONICS ENGINEERING

BLG252E – Object-Oriented Programming
2021 Spring

Lecturer: Asst. Prof. Dr. Tankut Akgül
T.A: Res. Asst. Halil Durmuş

Project 2

Mustafa İzzet Muştu – 040180702

Introduction

Aim of this project is to make a 2D shooter game by using SFML Library. There will be 3 type of objects; 2 soldiers, barrels and sandbags on the ground. Each object type has different textures, parameters and methods. However, they have common features which make them objects in the game. So, these common features will be defined in an Object class and other classes will inherit from that parent Object class because there is a “is” relationship.

There is also a Bullet class which inherited from Object class but instances of this class will be created when one of the players shoot and they will travel much faster than players. These Bullet objects will be created and handled in a friend class which is called BulletList. Object of this class will hold a linked list of Bullet objects and several operations like adding, deleting will be implemented in this class. Barrel and sandbag objects will stand still and won’t move. When a bullet collides with a sandbag it will be deleted from the linked list and won’t go further. However, when a bullet hits a barrel, both barrel and sandbag will be invisible (barrel will be invisible and bullet will be deleted).

The soldiers are expected to walk in direction which they are looking at and to stop when they collide the other objects. Soldiers also have 14 different textures for different states. Each state represents soldiers’ current situation like facing right, facing left etc. 14 different states will be implemented by looking state diagram in the project file. In addition, soldiers will have a score. When a bullet hits a soldier, other soldier’s score will increase and the soldier hit by bullet will spawn randomly in any other location (but not close where he dies) in the game ground. When one of the soldiers’ score becomes 10, game will end, there will be a text show up and it will say “Player X wins, start over? (Y/N)”. If one of the players want it, game will start from beginning. Figure 1 shows what game will look like when it starts.

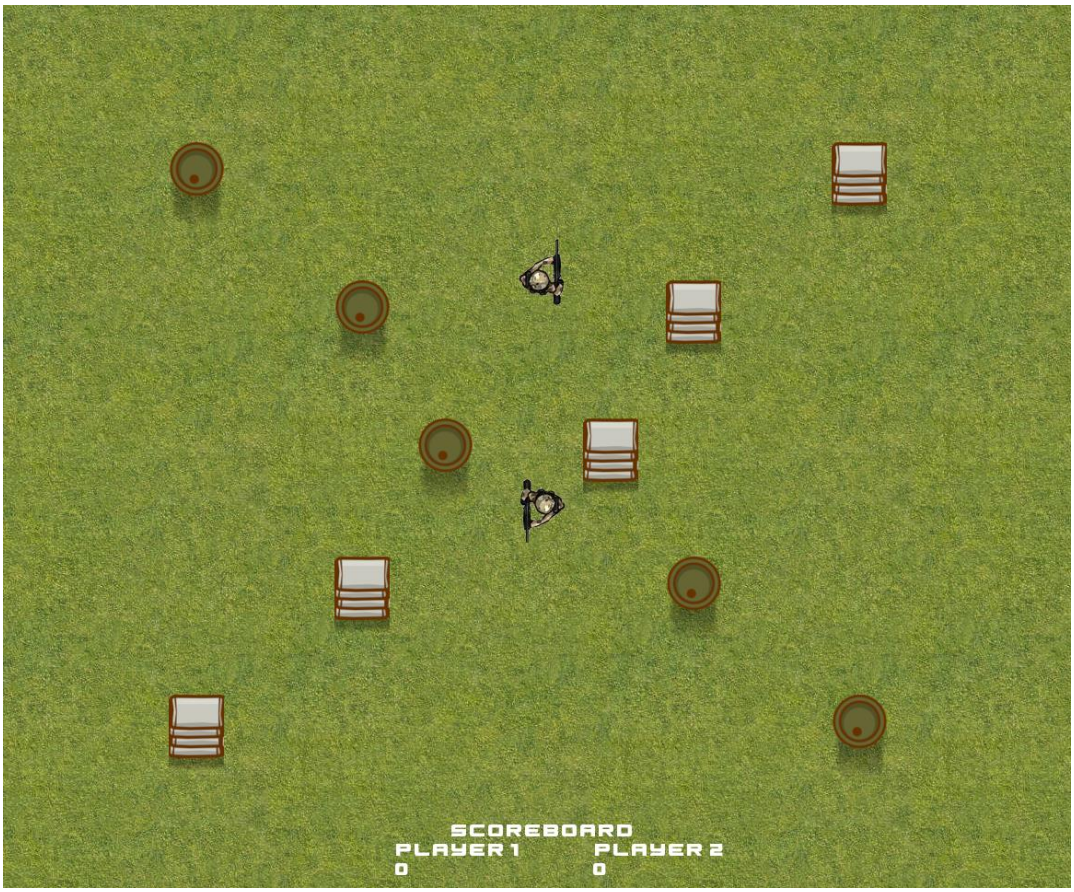

Figure 1

Implementation

class Object: This class represents visible objects in the game. Every game object (barrel, sandbag, bullet and player) will inherit from this class. This class is implemented as abstract so that instances of this class can't be created. There are 4 common attributes and 2 common methods for all subclasses. These are declared in this base class. Window pointer, Texture pointer, Sprite and Coord type variables are common attributes. Coord represents coordinate, Sprite represents sprite in the game. Window pointer is used to attach the object to game window. Texture pointer is declared as pointer because object textures differ and initialization are different; names of textures also differ. For example, player objects have 14 different textures but others have 1 texture. "init()" method is implemented as a pure virtual function so every subclass object must define this method. "paint()" method draws the game object in the game window and it is not overridden in the subclasses. Destructor is also defined as pure virtual and does nothing.

class Barrel: In addition to project 1, isVisible attribute, getter and setter functions for this attribute are added. When a bullet hits a barrel object, barrel becomes invisible.

class Sandbag: There are no change in this class with respect to project 1.

class Bullet: Speed and angle attributes are declared in addition to base object attributes. Speed can be chosen as we wish, however, angle is arranged that it can take only 4 values. Because, player can shoot only 4 direction. There is also a setter function for speed but not for angle. Inherited "init()" function is defined but it does nothing. Newly declared "init()" function of this class takes one more argument which is the state of the player. This new argument describes the angle of the bullet. "move()" function updates the bullet's position according to angle of the bullet. Destructors of barrel, sandbag, player and bullet classes deletes the textures accordingly. Lastly, BulletList class is declared as friend so that BulletList can access the next pointer of the bullet object.

class BulletList: Purpose of this class is to create a linked list of bullet objects and handle the related operations such as add, delete and update. These given operations are simple methods that adds a new bullet to linked list, deletes a bullet from linked list or deletes the entire linked list. "update()" method is also very simple, it iterates all the nodes in the linked list, calls "paint()" and "move()" methods for each bullet. "checkCollision()" method is a bit tricky. It does not return anything. It iterates through all the other object types for each bullet in the linked list. It also checks window boundaries for every bullet in the linked list. When a collision is detected between a bullet and sandbag, bullet is deleted from linked list. However, when a bullet hits a barrel object, not only the bullet but also the barrel is gone. Barrel object is not deleted, it becomes invisible. For the players, if there is collision between a bullet and a player, the player is spawned in a random location and other player gets his score incremented.

Logic of this collision function is kept very simple. It takes 2 coordinates from both the bullet and the object. Coordinates of the bullet are position (which is left top point) and the most left center (because thickness is 2 and it can be neglected) of the bullet. Other objects coordinates are the position, and right bottom point. When at least half of the bullet is inside the other object, there will be a collision. Actually, this method is not a good implementation because it does not give the correct result. For correct result, this function needs to check 4 corner points instead of 2 points to check 4 possible collision direction but in this way code becomes longer. And also, applied method runs smoothly and difference is not noticeable.

This collision logic is the logic which is used in player's collision detection method. For the attributes of this class, bullet pointer list is used as beginning pointer of linked list. RenderWindow pointer typed window and string typed path are used to pass argument to bullet methods.

class Player: Score attribute is added to this class in addition to project 1 and getter, setter methods for this attribute are added. Related to this attribute, "incrementScore()" method is also added. It just simply increments the score of the player by 1. There is a "startAgain()" function which very similar to "init()" function. It takes a position argument, arranges position and resets texture of the player. It is used when game restarts. For the randomly spawn of the player, "randomSpawn()" method is added. At first, it generates random coordinate numbers inside the window then it checks that the new coordinate is far from the current one. If the new one(position) is close to current one it generates new numbers and starts from beginning again. After, it iterates through all the other objects in the game ground and checks if there is a collision. If there is a collision, it goes to beginning. If there is not a collision, player is spawned in the new location.

5 new variables are added as attributes. These variables represent actions for walking and shooting. There is also a setter method added for them. These variables and method are used in the "update()" function of game class to make the player move or shoot.

class Game: BulletList pointer typed list is added to handle the bullet operations in the game. Text, Font typed variables are added to handle the scoreboard (also after game text). There is also a string variable added to store the string of the text object. Besides from these, "reset()" and "checkScore()" methods are new compared with project 1. Second one just checks scores of the players, if one of them is 10 it calls the first one. "reset()" function changes the string and text object for scoreboard. It moves the text object in the middle of the screen. It updates the game window and wait for a "Y" or "N" key to continue. If "Y" is pressed, it starts game from beginning. If "N" is pressed, game is closed.

In the "update()" method, implementation method for actions are changed to be able to handle multiple key pressed. Not only the "KeyPressed" events are checked but also "KeyReleased" events are checked in the event polling loop. When a players actions keys are pressed, related action variables are set true; when a players actions keys are released, related action variables are set false. At the end, players make actions if regarding variables are true. If-else statement is applied for each player so that he can move only one direction at a time.

int main(): No changes in this function. It is held very simple.

Fixes from first project: In first project, "setOrigin()" method is used unwittingly instead of "setPosition()" method for object sprites. So, parameters were opposite of they should have been. In this project, this problem is fixed. Also, there was a double check of game state. The game was controlled twice if it was open or not in the main, and in the "update()" function of the game so this is fixed.

Discussion

Biggest problem I encounter was the bullets initialization. When I create a bullet, it started in position that is not in the game window. Then I realized that I was not implementing correctly sprites position. I changed the method I called and the problem is solved. In addition to that, size of the program was increasing when a player shot but it was not decreasing when the bullet was deleted from linked list. Problem was the textures. Textures were not deleted

when the bullet deleted because texture is declared as pointer and defined with new operator so I should have called delete keyword in the object's destructor. I fixed this problem by doing that. Biggest issue but that could not be solved yet is the bullets movement. Movement is arranged that the new position is equal the current plus speed. However, it shouldn't be like that because when speed is very large, it can skip a game object without colliding the object. So, the bullet should pass from each pixel and shouldn't omit one regardless of the speed. To solve this problem, we can make new position equal the current plus one and do that action for the speed times.

Some screenshots from the game:

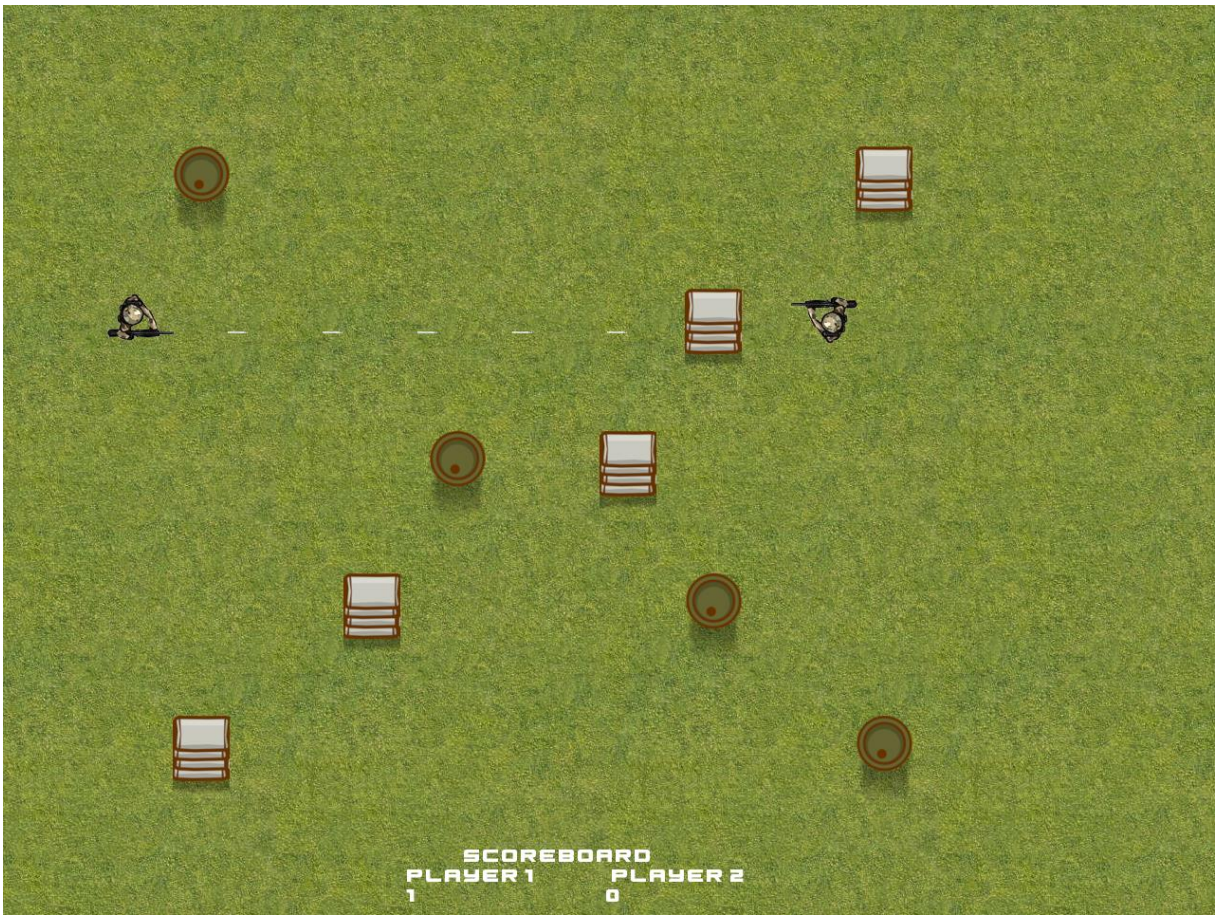

Figure 2

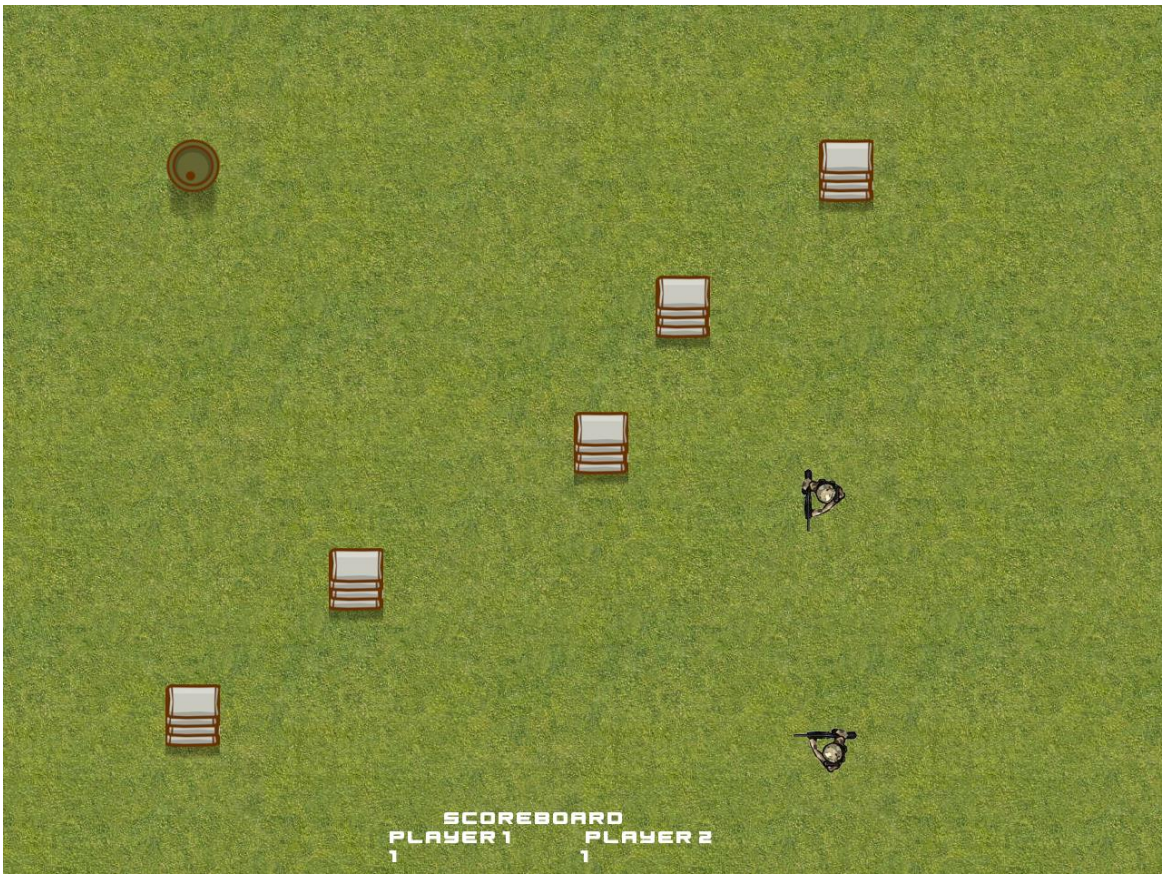

Figure 3

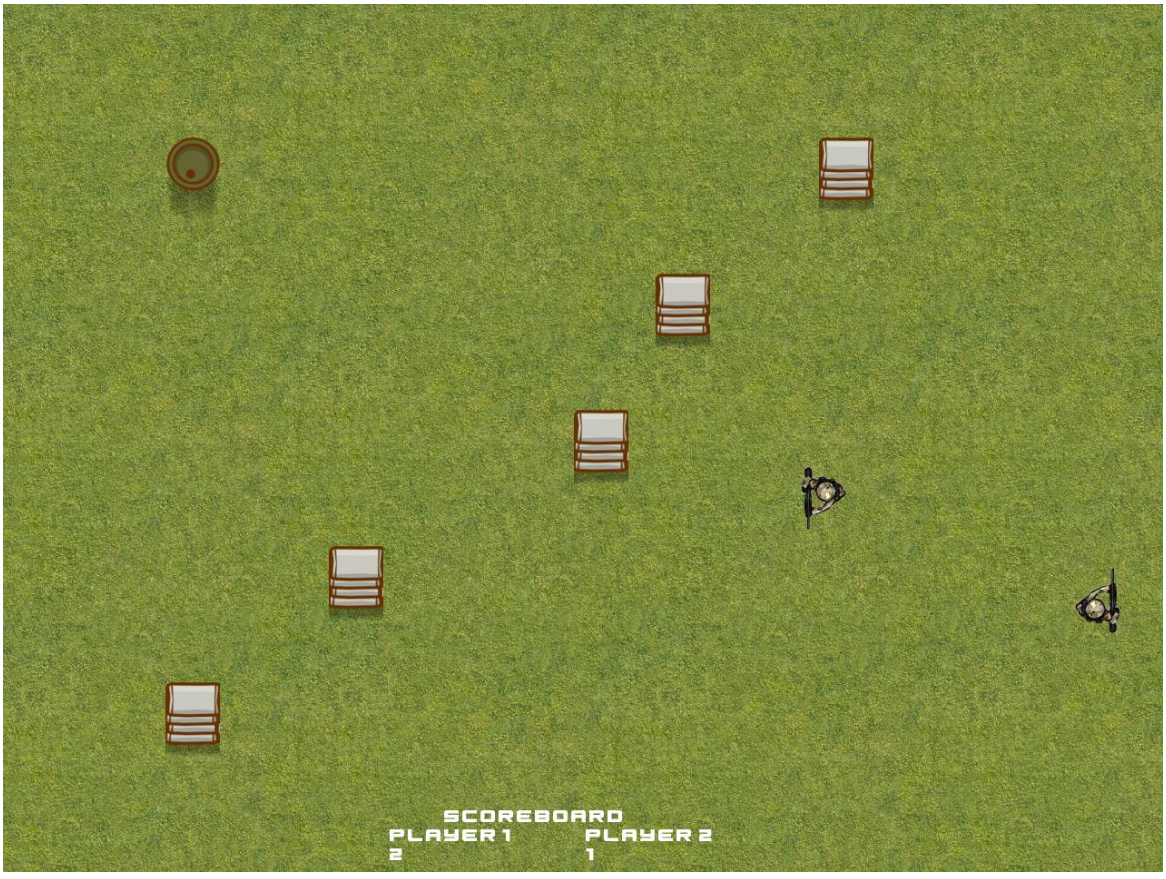

Figure 4

Figure 5